从Collection接口继承而来的一般成为：聚类（或 集合）；

从Map接口继承而来的一般称之为：映射。





排序:

Ordered的排列：按某种由具体情况决定的顺序排列。（后天决定的）

Sorted排列：按照天然的顺序排列，是由先天指定的，自身条件影响的。（不受外界影响，反应本质顺序。）

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Java:集合类性能分析  -集合接口：6个接口（短虚线表示），表示不同集合类型，是集合框架的基础。  -抽象类：5个抽象类（长虚线表示），对集合接口的部分实现。可扩展为自定义集合类。  -实现类：8个实现类（实线表示），对接口的具体实现。  2.Java容器类介绍  ① Java容器类都可以自动地调整自己的尺寸。  ② Collection 接口是一组允许重复的对象。  ③ Set 接口继承 Collection，不允许重复，使用自己内部的一个排列机制。  ④ List 接口继承 Collection，允许重复，以元素安插的次序来放置元素，不会重新排列。  ⑤ Map接口是一组成对的键－值对象，即所持有的是key-value pairs。Map中不能有重复的key。拥有自己的内部排列机制。  Java 2简化集合框架图  Java:集合类性能分析  3.Collection接口  基本操作  -增加元素add(Object obj); addAll(Collection c);  -删除元素 remove(Object obj); removeAll(Collection c);  -求交集 retainAll(Collection c);  Collection是最基本的集合接口，所有实现Collection接口的类都必须提供两个标准的构造函数：无参数的构造函数用于创建一个空的Collection，有一个 Collection参数的构造函数用于创建一个新的Collection，这个新的Collection与传入的Collection有相同类型的元素。  import java.util.ArrayList; import java.util.Arrays; import java.util.Collection; import java.util.Collections; import java.util.List; public class AddingGroups { public static void main(String[] args) {  Collection<Integer> collection = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4, 5));  Integer[] moreInts={6,7,8,9,10};  collection.addAll(Arrays.asList(moreInts));  for (Integer i : collection)  System.out.print(i + ","); } }  结果：  1,2,3,4,5,6,7,8,9,10,  这里展示了Collection接口的2个用法，首先，Collection构造函数接受另一个Collection（List）作为参数，使其初始化。接着，调用addAll()方法添加元素，注意，该方法只接受另一个Collection作为参数。  此外，必须注意，Collection接口不提供随机访问元素的get()方法。因为Collection包括Set，而Set自己维护内部顺序。如果想检查Collection中的元素，那就必须使用迭代器。  4.List接口  4.1 List接口  List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。  和下面要提到的Set不同，List允许有相同的元素。  除了具有Collection接口必备的iterator()方法外，List还提供一个listIterator()方法，返回一个 ListIterator接口，和标准的Iterator接口相比，ListIterator多了一些add()之类的方法，允许添加，删除，设定元素，还能向前或向后遍历。  4.2 LinkedList类  LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在 LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。此实现不是同步的。  4.3 ArrayList类  ArrayList实现了可变大小的数组。它允许所有元素，包括null。  size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。其他的方法运行时间为线性。  每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。此实现不是同步的。  5.Set接口  5.1 Set接口  Set具有和Collection完全一样的接口，没有任何额外的功能。它是一种不包含重复的元素的Collection，即任意的两个元素e1和e2都有e1.equals(e2)=false，Set最多有一个null元素。  很明显，Set的构造函数有一个约束条件，传入的Collection参数不能包含重复的元素。  请注意：必须小心操作可变对象（Mutable Object）。如果一个Set中的可变元素改变了自身状态导致Object.equals(Object)=true将导致一些问题。  5.2 HashSet  此类实现Set 接口，由哈希表（实际上是一个 HashMap 实例）支持。它不保证 set 的迭代顺序；特别是它不保证该顺序恒久不变。此类允许使用 null 元素。此类为基本操作提供了稳定性能，此实现不是同步的。  5.3 LinkedHashSet  具有可预知迭代顺序的Set 接口的哈希表和链接列表实现。此实现与HashSet的不同之处在于，它维护着一个运行于所有条目的双重链接列表。此链接列表定义了迭代顺序，即按照将元素插入到set中的顺序（插入顺序）进行迭代。注意，插入顺序不受在set中重新插入的元素影响。此实现不是同步的。  5.4 TreeSet  基于TreeMap的NavigableSet实现。使用元素的自然顺序对元素进行排序，或者根据创建set时提供的 Comparator 进行排序，具体取决于使用的构造方法。此实现为基本操作（add、remove 和 contains）提供受保证的 log(n) 时间开销。此实现不是同步的。  6.Map接口  请注意，Map没有继承Collection接口，Map提供key到value的映射。一个Map中不能包含相同的key，每个key只能映射一个 value。Map接口提供3种集合的视图，Map的内容可以被当作一组key集合，一组value集合，或者一组key-value映射。  6.1 WeakHashMap  以弱键实现的基于哈希表的Map。在WeakHashMap中，当某个键不再正常使用时，将自动移除其条目。更精确地说，对于一个给定的键，其映射的存在并不阻止垃圾回收器对该键的丢弃，这就使该键成为可终止的，被终止，然后被回收。丢弃某个键时，其条目从映射中有效地移除，因此，该类的行为与其他的Map实现有所不同。此实现不是同步的。  6.2 TreeMap  该映射根据其键的自然顺序进行排序，或者根据创建映射时提供的Comparator进行排序，具体取决于使用的构造方法。此实现不是同步的。  6.3 HashMap  基于哈希表的Map接口的实现。此实现提供所有可选的映射操作，并允许使用 null 值和 null 键。（除了非同步和允许使用null之外，HashMap 类与 Hashtable 大致相同。）此类不保证映射的顺序，特别是它不保证该顺序恒久不变。此实现不是同步的。  6.4 SortedMap  进一步提供关于键的总体排序的Map。该映射是根据其键的自然顺序进行排序的，或者根据通常在创建有序映射时提供的Comparator进行排序。对有序映射的collection 视图（由 entrySet、keySet 和 values 方法返回）进行迭代时，此顺序就会反映出来。要采用此排序方式，还需要提供一些其他操作（此接口是 SortedSet 的对应映射）。  7.集合类性能效率总结  注意，这里展示的类都是非线程安全的。如果需要考虑线程安全，应该使用ConcurrentMap,CopyOnWriteArrayList,CopyOnWriteArraySet等。   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 接口 | 实现类 | 保持插入顺序 | 可重复 | 排序 | 使用说明 | | List | ArrayList | Y | Y | N | 长于随机访问元素；但插入、删除元素较慢（数组特性）。 | | LinkedList | Y | Y | N | 插入、删除元素较快，但随即访问较慢（链表特性）。 | | Set | HashSet | N | N | N | 使用散列，最快的获取元素方法。 | | TreeSet | N | N | Y | 将元素存储在红-黑树数据结构中。默认为升序。 | | LinkedHashSet | Y | N | N | 使用散列，同时使用链表来维护元素的插入顺序。 | | Map | HashMap | N | N | N | 使用散列，提供最快的查找技术。 | | TreeMap | N | N | Y | 默认按照比较结果的升序保存键。 | | LinkedHashMap | Y | N | N | 按照插入顺序保存键，同时使用散列提高查找速度。 |   总结  ①如果涉及到堆栈，队列等操作，应该考虑用List。如果要进行大量的随机访问，应使用ArrayList；如果经常进行插入与删除操作，用使用LinkedList。  ②HashMap设计用来快速访问；而TreeMap保持“键”始终处于排序状态，所以没有HashMap快。LinkedHashMap保持元素插入的顺序，但是也通过散列提供了快速访问能力。  ③Set不接受重复元素。HashSet提供最快的查询速度，而TreeSet保持元素处于排序状态。LinkedHashSet以插入顺序保存元素。  ④对哈希表的操作，作为key的对象要正确重写equals和hashCode方法。  ⑤尽量返回接口而非实际的类型（针对抽象编程），如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，客户端代码不用改变。  ⑥程序中不应该使用过时的VectorHashtableStack。 |

Java集合框架（Java Collections Framework，JCF），类似与C++中的标准模板库（Standard Template Library，STL，大学时候想学结果倒也没怎么学）主要是对一些数据结构和相关算法的封装。

JCF的全家幅，如下图说示：

上图实在有点复杂，对初学者来说有点难度，我们先看个简单点的。

Collection

├List

│├LinkedList

│├ArrayList

│└Vector

│└Stack

└Set

Map

├Hashtable

├HashMap

└WeakHashMap

集合主要分为两个接口，Collection和Map接口。

Collection是最基本的集合接口，一个Collection代表一组对象，即Collection的元素（Elements）。

Collection提供关于集合的一些通用操作的接口，包括插入（add()方法）、删除（remove()方法）、

判断一个元素是不是其成员（contains()方法）、遍历（iterator()方法）等等

一些Collection允许相同的元素而另一些不行。一些能排序而另一些不行。如：

Set接口体现的是“无序集”的概念，它是不允许有重复元素出现的；List接口代表“有序集”，允许有重复元素；而Map接口则是“映射”关系

Java SDK不提供直接继承自Collection的类，它提供的类都是继承自Collection的“子接口”如List和Set。

有了“无序集”，“有序集”和“映射”，我们就可以定义各种各样的抽象数据结构了，如向量，链表，堆栈，哈希表，平衡二叉树等。

下面让我们来了解下各个子接口下实实在在的类。

如何遍历集合中的元素

使用该迭代即可逐一访问Collection中每一个元素。典型的用法如下：

Iterator it = collection.iterator(); // 获得一个迭代子

while(it.hasNext()) {

Object obj = it.next(); // 得到下一个元素

}

下面来介绍下关于List和Set的使用。

List是有序的Collection，使用此接口能够精确的控制每个元素插入的位置。

用户能够使用索引（元素在List中的位置，类似于数组下标）来访问List中的元素，这类似于Java的数组。

和下面要提到的Set不同，List允许有相同的元素。

除了具有Collection接口必备的iterator()方法外，List还提供一个listIterator()方法，返回一个ListIterator接口，

和标准的Iterator接口相比，ListIterator多了一些add()之类的方法，允许添加，删除，设定元素，还能向前或向后遍历。

实现List接口的常用类有LinkedList，ArrayList，Vector和Stack。

LinkedList类

LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在LinkedList的首部或尾部。

这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。

注意LinkedList没有同步方法。如果多个线程同时访问一个List，则必须自己实现访问同步。

一种解决方法是在创建List时构造一个同步的List：

List list = Collections.synchronizedList(new LinkedList(...));

ArrayList类

ArrayList实现了可变大小的数组。它允许所有元素，包括null。ArrayList没有同步。

size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。

其他的方法运行时间为线性。每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。

这个容量可随着不断添加新元素而自动增加，但是增长算法并没有定义。

当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。

和LinkedList一样，ArrayList也是非同步的（unsynchronized）。

Vector类

Vector非常类似ArrayList，但是Vector是同步的。由Vector创建的Iterator，虽然和ArrayList创建的Iterator是同一接口，

但是，因为Vector是同步的，当一个Iterator被创建而且正在被使用，另一个线程改变了Vector的状态（例如，添加或删除了一些元素），

这时调用Iterator的方法时将抛出ConcurrentModificationException，因此必须捕获该异常。

Stack 类

Stack继承自Vector，实现一个后进先出的堆栈。Stack提供5个额外的方法使得Vector得以被当作堆栈使用。

基本的push和pop方法，还有peek方法得到栈顶的元素，empty方法测试堆栈是否为空，search方法检测一个元素在堆栈中的位置。

Stack刚创建后是空栈。

Set接口

Set是一种不包含重复的元素的Collection，即任意的两个元素e1和e2都有e1.equals(e2)=false，Set最多有一个null元素。

很明显，Set的构造函数有一个约束条件，传入的Collection参数不能包含重复的元素。

Hashtable类

Hashtable继承Map接口，实现一个key-value映射的哈希表。任何非空（non-null）的对象都可作为key或者value。

添加数据使用put(key, value)，取出数据使用get(key)，这两个基本操作的时间开销为常数。

Hashtable通过initial capacity和load factor两个参数调整性能。通常缺省的load factor 0.75较好地实现了时间和空间的均衡。增大load factor可以节省空间但相应的查找时间将增大，这会影响像get和put这样的操作。

使用Hashtable的简单示例如下，将1，2，3放到Hashtable中，他们的key分别是”one”，”two”，”three”：

Hashtable numbers = new Hashtable();

numbers.put(“one”, new Integer(1));

numbers.put(“two”, new Integer(2));

numbers.put(“three”, new Integer(3));

要取出一个数，比如2，用相应的key：

Integer n = (Integer)numbers.get(“two”);

System.out.println(“two = ” + n);

由于作为key的对象将通过计算其散列函数来确定与之对应的value的位置，因此任何作为key的对象都必须实现hashCode和equals方法。

hashCode和equals方法继承自根类Object，如果你用自定义的类当作key的话，要相当小心，按照散列函数的定义，如果两个对象相同，

即obj1.equals(obj2)=true，则它们的hashCode必须相同，但如果两个对象不同，则它们的hashCode不一定不同，

如果两个不同对象的hashCode相同，这种现象称为冲突，冲突会导致操作哈希表的时间开销增大，（这个学过数据结构的人都应该知道吧）

所以尽量定义好的hashCode()方法，能加快哈希表的操作。

如果相同的对象有不同的hashCode，对哈希表的操作会出现意想不到的结果（期待的get方法返回null），要避免这种问题，

只需要牢记一条：要同时复写equals方法和hashCode方法，而不要只写其中一个。

Hashtable是同步的。

HashMap类

HashMap和Hashtable类似，不同之处在于HashMap是非同步的，并且允许null，即null value和null key。

但是将HashMap视为Collection时（values()方法可返回Collection），其迭代子操作时间开销和HashMap的容量成比例。

因此，如果迭代操作的性能相当重要的话，不要将HashMap的初始化容量设得过高，或者load factor过低。

他们之间又是怎么区分的呢？什么时候又要用到谁呢？

1. 侧重点：遍历 vs. 修改

遍历和修改似乎是一对矛盾，一个可以高效率插入删除元素的数据结构通常遍历的性能并不是最优。

于是JCF在这里根据用户的目标实现了两种定制的数据结构：哈希表（包括HashSet和HashMap）和平衡二叉树（包括TreeSet和TreeMap）。

由于可排序性是一种独特的要求，所以引入了SortedSet和SortedMap，它们分别是AbstractSet和AbstractMap的子接口，

而TreeSet和TreeMap又分别是他们的一种实现。熟悉数据结构的人可能比较了解，哈希表在进行插入、删除、查找这样的操作是很快的，

其时间复杂度是常数级O(1)；平衡二叉树虽然插入、删除操作比较麻烦（需要O(log n)的代价），但进行遍历和排序却很快。

选择完全在于用户的侧重点，但由于类型转换的方便性，通常我们用哈希表构造一个集合以后，再把它转换成相应的树集进行遍历，

以获得较好的效果。

2. 历史实现 vs. 新实现

历史实现（Legacy Implementations）是JCF的一个术语，准确的意义不是很清楚，但大致可以认为在Java 2（JDK 1.2）出现以前的老版本中JCF的一个雏形框架。

在Java 2以后，JCF才开始完善健壮起来，新实现中出现了一些新的类用于替代老版本中的成员，但由于种种原因，

老版本中很多类都代表了传统数据结构的精髓部分，以及一些安全原因，所以仍然被我们使用着。

Enumeration vs. Iterator

Enumeration是一个传统的集合遍历工具，在新的JCF中使用的是Iterator，Iterator同样具有遍历功能，

还包含一个remove()方法来删除当前得到的元素。

Dictionary vs. Map

Dictionary是一个现在已经被标记为deprecated的类，实现了老版本中的映射功能，现在已经完全被Map取代。

它们的区别是：Dictionary的key和value不能为null，但Map却允许空的关键字和值，

这一点直接影响到它们的后代：Hashtable和HashMap。

Vector vs. ArrayList

Vector和ArrayList是数组在JCF中的体现，还记得前面讲过的数组的缺点么？

Vector和ArrayList就是一种可以动态增长的数组。Vector是历史实现，它和ArrayList的主要区别在于，

Vector是同步集合（或者说是线程安全的），但ArrayList并不是同步的，由于同步需要花一定的代价，

所以ArrayList看起来要比Vector的存取访问效率更高。关于同步我们下面还将要谈到。

（我刚开始的时候，我们的项目里用的都是Vector ，其实是错误的，或则说性能不高的。因为那些项目涉及的都

是单线程，没有线程安全的概念，所以都应该使用ArrayList）这两者的比较我们在以后还会出现。这里只稍微提及下。

Hashtable vs. HashMap

Hashtable是Dictionary的子类，属于历史实现，而HashMap是Map的子类，是新实现。

它们的区别除了上面所说的key和value是否可以为空之外，也有同步的差别，Hashtable是同步的，

但HashMap不是。不过不要因为Hashtable是“老前辈”而瞧不起它哦，它的一个著名的子类Properties我们可是经常会用到的。

3. 同步 vs. 不同步

从上面的描述中我们似乎可以得出这么一个印象：历史实现好像都是同步的，但新实现中却没有。需要同步操作的理由是，

可能存在多个线程对同一个集合进行操作的情况：譬如一个线程正在对某集合进行遍历，但与此同时，

另一个线程又在对该集合进行插入或删除，那么第一个线程的遍历结果将是不可预测的，对于同步集合，

它将会抛出一个ConcurrentModificationException异常，JCF把这种机制成为“fail-fast”。

我们对比一下Vector和ArrayList的源代码就可以发现Vector的很多方法都是有synchronized关键字修饰的，但ArrayList没有。

4. 容易遗忘的工具：Collections和Arrays

在图1中右下角落里有两个类叫做Collections（注意，不是Collection！）和Arrays，这是JCF里面功能强大的工具，但初学者往往会忽视。按JCF文档的说法，这两个类提供了封装器实现（Wrapper Implementations）、数据结构算法和数组相关的应用。

想必大家不会忘记上面谈到的“折半查找”、“排序”等经典算法吧，Collections类提供了丰富的静态方法帮助我们轻松完成这些在数据结构课上烦人的工作：

binarySearch：折半查找。

sort：排序，这里是一种类似于快速排序的方法，效率仍然是O(n \* log n)，但却是一种稳定的排序方法。

reverse：将线性表进行逆序操作，这个可是从前数据结构的经典考题哦！

rotate：以某个元素为轴心将线性表“旋转”??哇，这个功能太酷了！

swap：交换一个线性表中两个元素的位置。

……

5. 泛型

目前我们了解的JCF的一个重要特征是：所有加入到集合当中的对象都将在表面上失去它们自己的特性，

而看上去仅仅只是一个Object对象而已，除非你把它强制类型转换成它们原来的对象。这一点很自然，集合嘛，对象的容器，

它容纳的是各种各样的对象，而不仅仅是某种特定类型的对象。J2SE 5.0出现以后，JCF开始引入泛型的特性，

譬如我们经常碰到这样的应用，就是把集合转换成特定的数组，虽然Collection有toArray()的方法，但可惜的是，

这个数组的所有元素都是Object类型的，我们通常的做法是用一个for循环把数组的每个元素都进行强制类型转换，

虽然可行，但看上去很笨拙，如果有了泛型，我们就可以预先指定要得到的类型，然后一次toArray就可以得到我们期望的数组，

里面的元素全部都是指定类型了。惭愧的是，我对5.0还不是太了解，具体可以参考J2SE 5.0的JCF文档

6，使用Vector还是ArrayList?

(1) Vector是同步的。这个类中的一些方法保证了Vector中的对象是线程安全的。而ArrayList则是异步的，

因此ArrayList中的对象并不是线程安全的。因为同步的要求会影响执行的效率，

所以如果你不需要线程安全的集合那么使用ArrayList是一个很好的选择，这样可以避免由于同步带来的不必要的性能开销。

(2)数据增长

从内部实现机制来讲ArrayList和Vector都是使用数组(Array)来控制集合中的对象。当你向这两种类型中增加元素的时候，

如果元素的数目超出了内部数组目前的长度它们都需要扩展内部数组的长度，Vector缺省情况下自动增长原来一倍的数组长度，

ArrayList是原来的50%,所以最后你获得的这个集合所占的空间总是比你实际需要的要大。

所以如果你要在集合中保存大量的数据那么使用Vector有一些优势，因为你可以通过设置集合的初始化大小来避免不必要的资源开销。

(3)使用模式

在ArrayList和Vector中，从一个指定的位置（通过索引）查找数据或是在集合的末尾增加、移除一个元素所花费的时间是一样的，

这个时间我们用O(1)表示。但是，如果在集合的其他位置增加或移除元素那么花费的时间会呈线形增长：O(n-i)，

其中n代表集合中元素的个数，i代表元素增加或移除元素的索引位置。为什么会这样呢？

以为在进行上述操作的时候集合中第i和第i个元素之后的所有元素都要执行位移的操作。这一切意味着什么呢？

这意味着，你只是查找特定位置的元素或只在集合的末端增加、移除元素，那么使用Vector或ArrayList都可以。

如果是其他操作，你最好选择其他的集合操作类。比如，LinkList集合类在增加或移除集合中任何位置的

元素所花费的时间都是一样的?O(1)，但它在索引一个元素的使用缺比较慢－O(i),其中i是索引的位置.

使用ArrayList也很容易，因为你可以简单的使用索引来代替创建iterator对象的操作。LinkList也会为每个插入的元素创建对象，

所有你要明白它也会带来额外的开销。

最后，在《Practical Java》一书中Peter Haggar建议使用一个简单的数组（Array）来代替Vector或ArrayList。

尤其是对于执行效率要求高的程序更应如此。因为使用数组(Array)避免了同步、额外的方法调用和不必要的重新分配空间的操作。

总结

如果涉及到堆栈，队列等操作，应该考虑用List，对于需要快速插入，删除元素，应该使用LinkedList，如果需要快速随机访问元素，

应该使用ArrayList。如果程序在单线程环境中，或者访问仅仅在一个线程中进行，考虑非同步的类，其效率较高，

如果多个线程可能同时操作一个类，应该使用同步的类。要特别注意对哈希表的操作，作为key的对象要正确复写equals和hashCode方法。

尽量返回接口而非实际的类型，如返回List而非ArrayList，这样如果以后需要将ArrayList换成LinkedList时，

客户端代码不用改变。这就是针对抽象编程。

Java提供了数种持有对象的方式，包括语言内置的Array，还有就是utilities中提供的容器类(container classes)，又称群集类(collection classes)。集合在java中非常重要，在讨论之前，先来看几个面试中的经典问题。

1 Collection 和 Collections的区别。

2 List, Set, Map是否继承自Collection接口。

3 ArrayList和Vector的区别。

4 HashMap和Hashtable的区别。

篇尾有答案，我们开始正题。

集合Collection接口

--Collection 是任何对象组，元素各自独立，通常拥有相同的套用规则。Set List由它派生。

基本操作

增加元素 add(Object obj); addAll(Collection c);

删除元素 remove(Object obj); removeAll(Collection c);

求交集 retainAll(Collection c);

访问/遍历集合元素的好办法是使用Iterator接口(迭代器用于取代Enumeration)

Public interface Iterator{

Public Boolean hasNext(0;

Public Object next(0;

Public void remove(0;

}

集set

--没有重复项目的集合

有三种特定类型的集可用

HashSet-基于散列表的集，加进散列表的元素要实现hashCode()方法

LinkedHashSet-对集迭代时，按增加顺序返回元素

TreeSet-基于（平衡）树的数据结构

清单List

--位置性集合。加进清单的元素可以加在清单中特定位置或加到末尾

有两个特定版本

ArrayList(数组表)-类似于Vector，都用于缩放数组维护集合。区别：

一.同步性:Vector是线程安全的，也就是说是同步的，而ArrayList是线程序不安全的，不是同步的

二.数据增长:当需要增长时,Vector默认增长为原来一培，而ArrayList却是原来的一半

LinkedList(链表)-是双向链表，每个节点都有两个指针指向上一节点和下一节点。

用在FIFO，用addList()加入元素 removeFirst()删除元素

用在FILO,用addFirst()/removeLast()

ListIterator提供双向遍历next() previous()，可删除、替换、增加元素

映射表Map

--用于关键字/数值对，像个Dictionary

处理Map的三种集合

关键字集KeySet()

数值集value()

项目集enrySet()

四个具体版本

HashMap-散列表的通用映射表

LinkedHashMap-扩展HashMap，对返回集合迭代时，维护插入顺序

WeakHashMap-基于弱引用散列表的映射表，如果不保持映射表外的关键字的引用，则内存回收程序会回收它

TreeMap-基于平衡树的映射表

Collections类，用于同步集合，还能改变集合只读方式的类

e.g.:

Map mp=new HashMap()

mp=Collections.synchronizedMap(mp); //生成线程安全的映射表

mp=Collections.unmodifiableMap(mp); //生成只读映射表

Comparable 自然顺序的排序类 Comparator 面向树的集合排序类

容器分类学(Container taxonomy)

集合接口： Collection List Set;Map Iterator ListIterator。

抽象类： AbstractCollection AbstractList AbstractSet AbstractMap AbstractSequentiaList。

Implementations

Hash TableResizable ArrayBalanced TreeLinked List

InterfacesSet HashSet TreeSet

List ArrayList LinkedList

Map HashMap TreeMap

老版本中的集合类型

Vector类

Vector，就是向量。一种异构的混合体，可以动态增加容量。对它的操作简要如下

比如我们有一个Vector: Vector myVec=new Vector(a\_Array.length)

取得vector的长度:myVec.size();

赋值：set(int position,Object obj) / setElementAt(Object obj, int position) –不支持动态增长

add(Object obj )/ addElement(Object obj) 在Vector末尾加入对象

e.g.：myVec.add(new a\_Array[0]);

取出元素：get(int position) / getElement(int position)

Stack类

是Vector的子类。就是数据结构里讲滥了的堆栈（这个词可简称栈，不要混淆于heap-堆）。后进先出的存取方式。

Stack()构造空栈

Empty()叛空

Search()检查堆栈是否有元素

Peek()取得栈顶元素

Pop()弹栈

Push()入栈

Enumeration接口

Dictionary类

字典。关键字/数值方式存取数据，如果映射没有此关键字，取回null。

Hashtable类

是Dictionary结构的具体实现。

面试题答案

Collection 和 Collections的区别。

Collections是个java.util下的类，它包含有各种有关集合操作的静态方法。

Collection是个java.util下的接口，它是各种集合结构的父接口。

List, Set, Map是否继承自Collection接口? List，Set是 Map不是

ArrayList和Vector的区别。

一.同步性:Vector是线程安全的，也就是说是同步的，而ArrayList是线程序不安全的，不是同步的

二.数据增长:当需要增长时,Vector默认增长为原来一培，而ArrayList却是原来的一半

HashMap和Hashtable的区别

一.历史原因:Hashtable是基于陈旧的Dictionary类的，HashMap是Java 1.2引进的Map接口的一个实现

二.同步性:Hashtable是线程安全的，也就是说是同步的，而HashMap是线程序不安全的，不是同步的

三.值：只有HashMap可以让你将空值作为一个表的条目的key或value

Vector的方法都是同步的(Synchronized),是线程安全的(thread-safe)，

而ArrayList的方法不是，由于线程的同步必然要影响性能，因此,ArrayList的性能比Vector好。

当Vector或ArrayList中的元素超过它的初始大小时,Vector会将它的容量翻倍,而ArrayList只增加50%的大小，这样,ArrayList就有利于节约内存空间。

Hashtable和HashMap

它们的性能方面的比较类似 Vector和ArrayList，比如Hashtable的方法是同步的,而HashMap的不是。

ArrayList和LinkedList

对 于处理一列数据项,Java提供了两个类ArrayList和LinkedList,ArrayList的内部实现是基于内部数组Object[],所以 从概念上讲,它更象数组，但LinkedList的内部实现是基于一组连接的记录,所以,它更象一个链表结构，所以,它们在性能上有很大的差别。

（1） 从上面的分析可知,在ArrayList的前面或中间插入数据时,你必须将其后的所有数据相应的后移,这样必然要花费较多时间，所以,当你的操作是在一列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用ArrayList会提供比较好的性能。

（2）而访问链表中的某个元素时,就必须从链表的一端开始沿着连接方向一个一个元素地去查找,直到找到所需的元素为止，所以,当你的操作是在一列数据的前面或中间添加或删除数据,并且按照顺序访问其中的元素时,就应该使用LinkedList了。

（3）如果在编程中,1，2两种情形交替出现,这时,你可以考虑使用List这样的通用接口,而不用关心具体的实现，在具体的情形下,它的性能由具体的实现来保证。

设置集合类的初始大小

在Java 集合框架中的大部分类的大小是可以随着元素个数的增加而相应的增加的，我们似乎不用关心它的初始大小,但如果我们考虑类的性能问题时,就一定要考虑尽可能地设置好集合对象的初始大小,这将大大提高代码的性能，比如,Hashtable缺省的初始大小为101,载入因子为0.75,即如果其中的元素个数超过 75个,它就必须增加大小并重新组织元素，所以,如果你知道在创建一个新的Hashtable对象时就知道元素的确切数目如为110,那么,就应将其初始 大小设为110/0.75=148,这样,就可以避免重新组织内存并增加大小。

特别要理解的：

Hashtable类

　　Hashtable继承Map接口，实现一个key-value映射的哈希表。任何非空（non-null）的对象都可作为key或者value。

　　添加数据使用put(key, value)，取出数据使用get(key)，这两个基本操作的时间开销为常数。

Hashtable 通过initial capacity和load factor两个参数调整性能。通常缺省的load factor 0.75较好地实现了时间和空间的 均衡。增大load factor可以节省空间但相应的查找时间将增大，这会影响像get和put这样的操作。

使用Hashtable的简单示例如下，将1，2，3放到Hashtable中，他们的key分别是”one”，”two”，”three”：

　　　　Hashtable numbers = new Hashtable();

　　　　numbers.put(“one”, new Integer(1));

　　　　numbers.put(“two”, new Integer(2));

　　　　numbers.put(“three”, new Integer(3));

　　要取出一个数，比如2，用相应的key：

　　　　Integer n = (Integer)numbers.get(“two”);

　　　　System.out.println(“two = ” + n);

　 　由于作为key的对象将通过计算其散列函数来确定与之对应的value的位置，因此任何作为key的对象都必须实现hashCode和equals方 法。hashCode和equals方法继承自根类Object，如果你用自定义的类当作key的话，要相当小心，按照散列函数的定义，如果两个对象相 同，即obj1.equals(obj2)=true，则它们的hashCode必须相同，但如果两个对象不同，则它们的hashCode不一定不同，如 果两个不同对象的hashCode相同，这种现象称为冲突，冲突会导致操作哈希表的时间开销增大，所以尽量定义好的hashCode()方法，能加快哈希 表的操作。

　　如果相同的对象有不同的hashCode，对哈希表的操作会出现意想不到的结果（期待的get方法返回null），要避免这种问题，只需要牢记一条：要同时复写equals方法和hashCode方法，而不要只写其中一个。

　　Hashtable是同步的。

HashMap类

　 　HashMap和Hashtable类似，不同之处在于HashMap是非同步的，并且允许null，即null value和null key。，但 是将HashMap视为Collection时（values()方法可返回Collection），其迭代子操作时间开销和HashMap的容量成比 例。因此，如果迭代操作的性能相当重要的话，不要将HashMap的初始化容量设得过高，或者load factor过低。

LinkedList类

　 　LinkedList实现了List接口，允许null元素。此外LinkedList提供额外的get，remove，insert方法在 LinkedList的首部或尾部。这些操作使LinkedList可被用作堆栈（stack），队列（queue）或双向队列（deque）。

　　注意LinkedList没有同步方法。如果多个线程同时访问一个List，则必须自己实现访问同步。一种解决方法是在创建List时构造一个同步的List：

　　　　List list = Collections.synchronizedList(new LinkedList(...));

ArrayList类

　　ArrayList实现了可变大小的数组。它允许所有元素，包括null。ArrayList没有同步。

size，isEmpty，get，set方法运行时间为常数。但是add方法开销为分摊的常数，添加n个元素需要O(n)的时间。其他的方法运行时间为线性。

　 　每个ArrayList实例都有一个容量（Capacity），即用于存储元素的数组的大小。这个容量可随着不断添加新元素而自动增加，但是增长算法并 没有定义。当需要插入大量元素时，在插入前可以调用ensureCapacity方法来增加ArrayList的容量以提高插入效率。

　　和LinkedList一样，ArrayList也是非同步的（unsynchronized）。

Vector类

　 　Vector非常类似ArrayList，但是Vector是同步的。由Vector创建的Iterator，虽然和ArrayList创建的 Iterator是同一接口，但是，因为Vector是同步的，当一个Iterator被创建而且正在被使用，另一个线程改变了Vector的状态（例 如，添加或删除了一些元素），这时调用Iterator的方法时将抛出ConcurrentModificationException，因此必须捕获该 异常。

Array是数组,不在集合框架范畴之内,一旦选定了,它的容量大小就不能改变了,所以通常在编程中不选用数组来存放.

集合 :

集合对象：用于管理其他若干对象的对象

数组：长度不可变

List: 有顺序的，元素可以重复

遍历：for 迭代

排序：Comparable Comparator Collections.sort()

ArrayList：底层用数组实现的List

特点：查询效率高，增删效率低 轻量级 线程不安全

LinkedList：底层用双向循环链表 实现的List

特点：查询效率低，增删效率高

Vector: 底层用数组实现List接口的另一个类

特点：重量级，占据更多的系统开销 线程安全

Set：无顺序的，元素不可重复（值不相同）

遍历：迭代

排序：SortedSet

HashSet：采用哈希算法来实现Set接口

唯一性保证：重复对象equals方法返回为true

重复对象hashCode方法返回相同的整数

不同对象 哈希码 尽量保证不同（提高效率）

SortedSet：对一个Set排序

TreeSet：在元素添加的同时，进行排序。也要给出排序规则

唯一性保证：根据排序规则，compareTo方法返回为0，就可以认定两个对象中有一个是重复对象。

Map：元素是键值对 key：唯一，不可重复 value：可重复

遍历：先迭代遍历key的集合，再根据key得到value

HashMap:轻量级 线程不安全 允许key或者value是null

Hashtable：重量级 线程安全 不允许key或者value是null

Properties：Hashtable的子类，key和value都是String

SortedMap：元素自动对key排序

TreeMap：

集合是指一个对象可以容纳了多个对象（不是引用），这个集合对象主要用来管理维护一系列相似的对象。

集合接口类层次 :

位于package java.util.\*;

Collection

↑

|ˉˉˉˉˉˉ|

Set List Map

↑ ↑

| |

SortedSet SortedMap

1) Set: 集合类中不允许有重复对象;

2) SortedSet: 和Set接口同，但元素按升序排列;

3) List: 元素加载和移出时按照顺序，可以保存重复对象。

4) Map: (key-value对)存储了唯一关键字辨识和对应的值。

5) SortedMap: 和Map类同，但对象按他们关键字的升序排列。

集合类层次 :

（注：JAVA1.5对JAVA1.4的最大改进就是增加了对范型的支持）

Collection

↑

|ˉˉˉˉˉˉ|

HashSet LinkedList Hashtable

(Set) Vector, ArrayList Hashmap

(List) (Map)

↑ ↑

| |

TreeSet TreeMap

(SortedSet) (SortedMap)

Collection接口的方法：

add(Object o)

addAll(Collection c)

contains(Object o)

containsAll(Collection c)

remove(Object o)

removeAll(Collection c)

clear()

equals(Object o)

isEmpty()

iterator()

size()

toArray()

toArray(Object[] o)

五个最常用的集合类之间的区别和联系:

1．ArrayList: 元素单个，效率高，多用于查询

2．Vector: 元素单个，线程安全，多用于查询

3．LinkedList:元素单个，多用于插入和删除

4．HashMap: 元素成对，元素可为空

5．HashTable: 元素成对，线程安全，元素不可为空

ArrayList

底层是Object数组，所以ArrayList具有数组的查询速度快的优点以及增删速度慢的缺点。

而在LinkedList的底层是一种双向循环链表。在此链表上每一个数据节点都由三部分组成：前指针（指向前面的节点的位置），数据，后指针（指向后面的节点的位置）。最后一个节点的后指针指向第一个节点的前指针，形成一个循环。

双向循环链表的查询效率低但是增删效率高。

ArrayList和LinkedList在用法上没有区别，但是在功能上还是有区别的。

LinkedList

经常用在增删操作较多而查询操作很少的情况下：队列和堆栈。

队列：先进先出的数据结构。

栈：后进先出的数据结构。

注意：使用栈的时候一定不能提供方法让不是最后一个元素的元素获得出栈的机会。

Vector

（与ArrayList相似，区别是Vector是重量级的组件，使用使消耗的资源比较多。）

结论：在考虑并发的情况下用Vector（保证线程的安全）。

在不考虑并发的情况下用ArrayList（不能保证线程的安全）。

java.util.stack（stack即为堆栈）的父类为Vector。可是stack的父类是最不应该为Vector的。因为Vector的底层是数组，且Vector有get方法（意味着它可能访问到并不属于最后一个位置元素的其他元素，很不安全）。

对于堆栈和队列只能用push类和get类。

Stack类以后不要轻易使用。

实现栈一定要用LinkedList。

（在JAVA1.5中，collection有queue来实现队列。）

Set-HashSet实现类：

遍历一个Set的方法只有一个：迭代器（interator）。

HashSet中元素是无序的（这个无序指的是数据的添加顺序和后来的排列顺序不同），而且元素不可重复。

在Object中除了有finalize()，toString()，equals()，还有hashCode()。

HashSet底层用的也是数组。

当向数组中利用add(Object o)添加对象的时候，系统先找对象的hashCode：

int hc=o.hashCode(); 返回的hashCode为整数值。

Int I=hc%n;（n为数组的长度），取得余数后，利用余数向数组中相应的位置添加数据，以n为6为例，如果I=0则放在数组a[0]位置，如果I=1,则放在数组a[1]位置。如果equals()返回的值为true，则说明数据重复。如果equals()返回的值为false，则再找其他的位置进行比较。这样的机制就导致两个相同的对象有可能重复地添加到数组中，因为他们的hashCode不同。

如果我们能够使两个相同的对象具有相同hashcode，才能在equals()返回为真。

在实例中，定义student对象时覆盖它的hashcode。

因为String类是自动覆盖的，所以当比较String类的对象的时候，就不会出现有两个相同的string对象的情况。

现在，在大部分的JDK中，都已经要求覆盖了hashCode。

结论：如将自定义类用hashSet来添加对象，一定要覆盖hashcode()和equals()，覆盖的原则是保证当两个对象hashcode返回相同的整数，而且equals()返回值为True。

如果偷懒，没有设定equals()，就会造成返回hashCode虽然结果相同，但在程序执行的过程中会多次地调用equals()，从而影响程序执行的效率。

我们要保证相同对象的返回的hashCode一定相同，也要保证不相同的对象的hashCode尽可能不同（因为数组的边界性，hashCode还是可能相同的）。

例子：

public int hashCode(){

return name.hashcode()+age;

}

这个例子保证了相同姓名和年龄的记录返回的hashCode是相同的。

使用hashSet的优点：

hashSet的底层是数组，其查询效率非常高。而且在增加和删除的时候由于运用的hashCode的比较开确定添加元素的位置，所以不存在元素的偏移，所以效率也非常高。因为hashSet查询和删除和增加元素的效率都非常高。

但是hashSet增删的高效率是通过花费大量的空间换来的：因为空间越大，取余数相同的情况就越小。HashSet这种算法会建立许多无用的空间。

使用hashSet类时要注意，如果发生冲突，就会出现遍历整个数组的情况，这样就使得效率非常的低。

1.1.4. 比较

Collections类（工具类―――全是static 方法）

Public static int binarySearch(List list,Object key)

Public static void Sort(List list,Comparator com)

Public static void sort(List list)

方法一：

Comparator接口

Int compare(Object a,Object b)

Boolean equals(Object o)

例子：

import java.util.\*;

public class Test {

public static void main(String[] arg) {

ArrayList al = new ArrayList();

Person p1 = new Person("dudi");

Person p2 = new Person("cony");

Person p3 = new Person("aihao");

al.add(p1);

al.add(p2);

al.add(p3);

Collections.sort(al,p1);

for(Iterator it = al.iterator();it.hasNext();){

Person p = (Person)it.next();

System.out.println(p.name);

}

}

}

class Person implements java.util.Comparator

{

public String name;

public Person(String name){

this.name = name;

}

public int compare(Object a,Object b){

if(a instanceof Person&&b instanceof Person){

Person pa = (Person)a;

Person pb = (Person)b;

return pa.name.compareTo(pb.name);

}

return 0;

}

public boolean equals(Object a){return true;}

}

方法二

Java.lang.Comparable

Public int compareTo(Object o)

Class Person implements java.lang.Comparable{

Public int compareTo(Object o){

Comparable c1=(Comparable)this;

Comparable c2=(Comparable)o;

Return c1.name.compareTo(c2.name );

}

}

……………………………….

}